**Sprawozdanie z Projektu**

Autor: Stępień Mikołaj 269274

Kurs: Podstawy Podejmowania Decyzji

Grupa: Środa 11:15 – 12:45

**Wprowadzenie do problemu:**

Futoushiki ( z jap. 不等式 – nierówność matematyczna) jest to gra matematyczna prowadzona na kwadracie podzielonym na równe części, w której to gracz ma za zadanie wypełnić całą planszę liczbami od 1 do N, gdzie N to ilość wydzielonych kratek na krawędź kwadratu, stosując się przy tym do założonych nań ograniczeń w postaci:

Liczb startowych – gracz musi uwzględnić liczby wypełnione przed rozpoczęciem rozgrywki.

Zasad tzw. „Kwadratu łacińskiego” z ang. „Latin square”, czyli na jeden rząd i kolumnę może przypadać co najwyżej jedna unikalna cyfra.

Nierówności – Jest to charakterystyczny dla tej gry zestaw znaków większości pomiędzy kratkami, wymuszająca na graczu przypisanie jednej z sąsiadujących kratek wartości większej, bądź mniejszej.

Obraz zawierający kwadrat, wzór, diagram, Prostokąt

Opis wygenerowany automatycznie

Rysunek - Przykładowa plansza do gry w Fudoushiki.

Niektóre zestawy nierówności, oraz cyfr startowych są z góry niemożliwe do wypełnienia, np.: Znak większości przyłączony do kratki z 1 nie ma prawa zostać wypełniony żadną możliwą liczbą.

W sytuacji w której plansza jest niemożliwa do zrealizowania, za najlepsze uznajemy rozwiązanie zapełniające jak największą część planszy.

W tej pracy pole niewypełnione będzie oznaczane cyfrą 0.

W tym projekcie zająłem się odtwarzaniem tej gry, tworząc nie zawsze możliwe zestawy nierówności, oraz cyfr startowych, szukając jak najlepszej kombinacji.

**Dane, zmienne decyzyjne i ograniczenia:**

W dalszej część dla usprawnienia zapisu pojawią się następujące symbole:

Zmienne konfiguracji startowej:

* n – rozmiar planszy do gry
* r – ilość nierówności
* R – struktura danych składająca się z zagnieżdżonych list, zawierające informację o nierównościach na planszy
* l – ilość liczb startowych
* L - struktura danych składająca się z zagnieżdżonych list, zawierające informację o liczbach wpisanych na start do planszy

Symbole dla zmiennych decyzyjnych:

* X – macierz zawierająca liczby od 0 do N, będąca reprezentacją pola do gry, składająca się z elementów x.
* Z – macierz boolowska, będąca reprezentacją 0 występujących w macierzy rozwiązania, składająca się z elementów z.

Funkcja Celu F:

Szukamy rozwiązania jak najbardziej możliwie kompletnego, czyli zawierającego jak najmniejszą ilość zer:

Ograniczenia dla naszego problemu optymalizacji:

Na planszy nie ma prawa wystąpić liczba mniejsza od 0:

Na planszy nie ma prawa wystąpić liczba większa od n:

Ograniczenie wynikające z nierówności z uwzględnieniem wyjątku dla pustego pola:

Ograniczenie wynikające z istniejących na starcie liczb:

Ograniczenia kwadratu łacińskiego tj. liczba może wystąpić co najwyżej raz na kolumnę i rząd, pod warunkiem, że nie jest zerem:

Dla kolumn:

Dla rzędów:

Kod w CPLEX:

int n = 7; // Rozmiar planszy

int r = 18; // Liczba nierówności

int l = 4; // Liczba liczb startowych

int R[1..r][1..2][1..2] = // struktura danych zawierające nierownosci

[[[4, 2], [5, 3]], [[4, 6], [3, 6]],

[[5, 5], [4, 4]], [[3, 7], [2, 6]],

[[4, 3], [3, 3]], [[2, 1], [3, 1]],

[[2, 1], [2, 2]], [[2, 7], [1, 7]],

[[7, 3], [7, 4]], [[3, 4], [2, 3]],

[[4, 7], [5, 6]], [[7, 4], [7, 5]],

[[6, 4], [5, 5]], [[6, 6], [5, 7]],

[[2, 2], [3, 1]], [[4, 3], [3, 4]],

[[3, 5], [2, 6]], [[4, 2], [4, 1]]];

int L [1..L][1..2][1..2] = // struktura danych zawierające liczy startowe

[[[5, 6], [2, 2]], [[4, 1],

[3, 3]], [[6, 6], [5, 5]],

[[7, 3], [2, 2]]];

dvar int X[1..n][1..n]; // Plansza

dvar boolean Z[1..n][1..n];

dexpr int F = sum(i in 1..n) sum(j in 1..n) Z[i][j];

minimize ilosc\_zer; // Minimalizujemy zera

subject to {

forall(i in 1..n)

forall(j in 1..n)

X[i][j] >= 0; // Większe niż 0

forall(i in 1..n)

forall(j in 1..n)

X[i][j] <= n; //Mniejsze niż n

forall(i in 1..r)

(X[R [i][1][1]][R [i][1][2]] != 0) => // Jeżeli to nie jest zerem

(X[R [i][1][1]][R [i][1][2]]

>= X[R [i][2][1]][R [i][2][2]]); //Uwzglednienie nierownosci

//Uwzglednienie podanych na start liczb

forall(i in 1..l) X[L [i][1][1]][L [i][1][2]] == L[i][2][1];

// Kwadrat lacinski

forall(l in 1..n)

forall(i in 1..n)

sum(j in 1..n)

(X[i][j] == l) <= 1; // Jedna cyfra na rząd

forall(l in 1..n)

forall(i in 1..n)

sum(j in 1..n)

(X[j][i] == l) <= 1; // Jedna cyfra na kolumnę

forall(i in 1..n)

forall(j in 1..n)

(X[i][j] == 0) => (Z[i][j] == 1); // Tworzenie macierzy zer

}

**Metaheurystyka**

Do wykonania tego zadania postanowiłem użyć techniki tzw. Symulowanego rozżarzania, poszukując w ten sposób wyników gorszych, lecz o wiele lżejszych obliczeniowo i szybszych w uzyskaniu.

Technika ta opiera się na koncepcie „temperatury” która kontroluje prawdopodobieństwo przyjęcia rozwiązania gorszego. W ten sposób algorytm, może unikać minimów lokalnych.

W przypadku mojego rozżarzania przyjęto następujące atrybuty:

- Temperatura początkowa, określa, jaka będzie startowa temperatura w naszym algorytmie.

- Obecna temperatura w algorytmie.

– określa zmianę temperatury przy następnym kroku.

Gdzie: – krok następny, - krok obecny

S – warunek stopu

Algorytm działa tak długo, aż temperatura obecna nie będzie mniejsza lub równa warunkowi stopu.

Samo działanie algorytmu tworzącego rozwiązanie losowe naszego futoushiki wygląda następująco:

1. Wygeneruj planszę i nałóż na nią liczby startowe.
2. Zaczynając od górnego prawego rogu planszy. Iterując po każdym pustym polu planszy:
   1. Rozpatrz możliwych „kandydatów” tj. liczby które mogą zostać wpisane w danym miejscu.

Obraz zawierający diagram, kwadrat, design, wzór

Opis wygenerowany automatycznie

Rysunek 2- krok pierwszy

Przykładowo dla kroku 1 możliwi kandydaci dla pola (1,1) to {1, 2, 4}, 3 odpada ze względu na warunek kwadratu łacińskiego.

* 1. Wylosuj jednego z kandydatów. Zapisz do listy DECYZJE podjętą decyzje w formacie:

[ nr decyzji, pole, [kandydaci], wybrany]

Dla kroku 1 z rysunku 2 będzie to przykładowo:

[ 1, (1, 1), [1, 2, 4], 2]

W sytuacji, gdy lista kandydatów jest pusta, wstaw 0.

W ten sposób otrzymujemy wylosowaną konfigurację startową, oraz listę podjętych decyzji w tej konfiguracji.

Aby otrzymać rozwiązanie sąsiednie do obecnie akceptowanej konfiguracji:

1. Z listy DECYZJE poprzedniej konfiguracji wylosuj jedną, gdzie lista kandydatów jest dłuższa niż jeden element.
2. Stwórz nową listę N\_DECYZJE, która będzie zawierała wszystkie decyzje podjęte przed tą wybraną, oraz wybraną decyzje, ze zmienionym elementem „wybrany”.

Przykładowo:

Dla DECYZJE wylosowano rozwiązanie sąsiednie [ 4, (2, 1), [1, 2, 3], 2]:

DECYZJE:

[ 1, (1, 1), [1, 2, 4], 2]

[ 2, (1, 2), [1], 1]

[ 3, (1, 3), [4], 4]

[ 4, (2, 1), [1, 2, 3], 1]

[ 5, (2, 2), [2, 3, 4], 4]

[ 6, (2, 3), [2, 3], 2]

[ 7, (2, 4), [], 0]

N\_DECYZJE:

[ 1, (1, 1), [1, 2, 4], 2]

[ 2, (1, 2), [1], 1]

[ 3, (1, 3), [4], 4]

[ 4, (2, 1), [1, 2, 3], 2]

…. Dalej wedle algorytmu zapełniania.

W ten sposób powstaje nam rozwiązanie sąsiednie.

Rozwiązania sąsiednie będą porównywane pod względem funkcji celu F , którą była liczba zer w rozwiązaniu.

W przypadku gdy F nowe będzie mniejsze od F starego, przyjmujemy rozwiązanie.

W przypadku gdy F nowe jest większe of F starego, przyjęcie nowego rozwiązania ma prawdopodobieństwo określone wzorem:

Przez całą funkcję zapamiętujemy najlepszy wynik, oraz listę decyzji do niego dążącą.

Fragment kodu w python:

"Wsadzanie liczb startowych"  
for liczba in liczby:  
 plansza[liczba[0][0] - 1 ][liczba[0][1] - 1] = liczba[1][0]  
  
plansza\_start = plansza  
  
decyzje = []  
akceptowane\_decyzje = []  
d = 1  
  
czas\_poczotkowy = time.time()  
# Uzupełnij losowo poczatkowa orientacje  
for i in range(N):  
 for j in range(N):  
 if plansza[i][j] is None:  
 kandydaci = wykreuj\_kandydatow(plansza, [i, j], nierownosci)  
 wybor = random.choice(kandydaci)  
 plansza[i][j] = wybor  
 decyzje.append([d, [i, j], kandydaci, wybor])  
 d += 1  
  
  
#Akceptujemy pierwszą odpowiedz  
najlepsze\_decyzje = decyzje  
obecny\_l = wylicz\_zera(akceptowane\_decyzje)  
najlepszy\_l = wylicz\_zera(najlepsze\_decyzje)  
  
  
# Rozżarzanie Symulowane  
T\_poczotkowa = 2000  
T\_obecna = T\_poczotkowa  
alfa = 0.95  
warunek\_stopu = 1  
  
while T\_obecna >= warunek\_stopu:  
 n\_roz = wylosuj\_rozwiaznie\_sasiednie(decyzje)  
 n\_plansza, n\_decyzje = stworz\_plansze\_na\_podstawie\_decyzji(plansza\_start, n\_roz)  
 nowy\_l = wylicz\_zera(n\_decyzje)  
  
  
 if nowy\_l < obecny\_l:  
 decyzje = n\_decyzje  
 obecny\_l = nowy\_l  
 if nowy\_l < najlepszy\_l:  
 najlepsze\_decyzje = n\_decyzje  
 najlepszy\_l = nowy\_l  
  
 else:  
 wagi = [np.exp( -1 \* ( (obecny\_l - nowy\_l) / T\_obecna) ), 1 - np.exp( -1 \* ( (obecny\_l - nowy\_l) / T\_obecna) )]  
 wartosci = [1, 0]  
 wybnik = random.choices(wartosci, wagi)[0]  
 if wybnik == 1:  
 decyzje = n\_decyzje  
 obecny\_l = nowy\_l  
  
 T\_obecna = T\_obecna \* alfa

**Wyniki**

Po przeprowadzeniu serii 25 testów dla generowanych łamigłówek futoushiki przy następujących ustawieniach startowych:

n = 7

r = 18

l = 4

Dla rozżarzania o ustawieniach:

= 2000

= 0,95

S = 1

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Czas SA | Czas CP | F SA | F CP | różnica czasowa | różnica zer | różnica czasu / różnice zer |
| Próba numer: 1 | 0,056877 | 5,85 | 6 | 1 | 5,793122625 | 5 | 1,158624525 |
| Próba numer: 2 | 0,037903 | 5,96 | 4 | 0 | 5,922097168 | 4 | 1,480524292 |
| Próba numer: 3 | 0,037899 | 8,49 | 4 | 0 | 8,452101221 | 4 | 2,113025305 |
| Próba numer: 4 | 0,04288 | 9,41 | 7 | 1 | 9,36712018 | 6 | 1,561186697 |
| Próba numer: 5 | 0,038896 | 8,33 | 6 | 0 | 8,291104393 | 6 | 1,381850732 |
| Próba numer: 6 | 0,036901 | 7,68 | 7 | 1 | 7,643098526 | 6 | 1,273849754 |
| Próba numer: 7 | 0,036898 | 7,29 | 4 | 2 | 7,253102102 | 2 | 3,626551051 |
| Próba numer: 8 | 0,036901 | 7,54 | 5 | 2 | 7,503099241 | 3 | 2,50103308 |
| Próba numer: 9 | 0,03832 | 10,06 | 4 | 0 | 10,02167994 | 4 | 2,505419984 |
| Próba numer: 10 | 0,04117 | 12,47 | 5 | 1 | 12,42883012 | 4 | 3,10720753 |
| Próba numer: 11 | 0,036902 | 18,81 | 4 | 0 | 18,77309829 | 4 | 4,693274572 |
| Próba numer: 12 | 0,038897 | 3,46 | 5 | 1 | 3,421103439 | 4 | 0,85527586 |
| Próba numer: 13 | 0,03391 | 9,6 | 5 | 1 | 9,566089964 | 4 | 2,391522491 |
| Próba numer: 14 | 0,043881 | 9,15 | 6 | 0 | 9,106118822 | 6 | 1,51768647 |
| Próba numer: 15 | 0,041855 | 14,72 | 5 | 0 | 14,67814514 | 5 | 2,935629028 |
| Próba numer: 16 | 0,047871 | 4,8 | 5 | 0 | 4,752128887 | 5 | 0,950425777 |
| Próba numer: 17 | 0,036901 | 12,55 | 4 | 0 | 12,51309876 | 4 | 3,128274691 |
| Próba numer: 18 | 0,04192 | 10,86 | 5 | 0 | 10,81807958 | 5 | 2,163615915 |
| Próba numer: 19 | 0,039892 | 4,88 | 5 | 1 | 4,840108042 | 4 | 1,21002701 |
| Próba numer: 20 | 0,037899 | 8,02 | 6 | 2 | 7,982101221 | 4 | 1,995525305 |
| Próba numer: 21 | 0,038897 | 5,8 | 3 | 0 | 5,761102962 | 3 | 1,920367654 |
| Próba numer: 22 | 0,037476 | 8,65 | 5 | 1 | 8,612524414 | 4 | 2,153131104 |
| Próba numer: 23 | 0,037901 | 7,4 | 6 | 1 | 7,362099314 | 5 | 1,472419863 |
| Próba numer: 24 | 0,041858 | 7,83 | 5 | 1 | 7,788142281 | 4 | 1,94703557 |
| Próba numer: 25 | 0,068816 | 15,54 | 5 | 1 | 15,47118429 | 4 | 3,867796073 |
|  |  |  |  |  |  |  |  |
| średnia | 0,041181 | 9,006 | 5,04 | 0,68 | 8,964819237 | 4,36 | 2,156451213 |
| mediana | 0,038896 | 8,33 | 5 | 1 | 8,291104393 | 4 | 1,995525305 |
| minimum | 0,03391 | 3,46 | 3 | 0 | 3,421103439 | 2 | 0,85527586 |
| maksimum | 0,068816 | 18,81 | 7 | 2 | 18,77309829 | 6 | 4,693274572 |

Doszedłem do następujących wyników:

Wynika z nich, że dla przyjętych przeze mnie warunków początkowych dla symulowanego wyżarzania

problem rozwiązuje się o wiele szybciej niż, dla Solvera, jest on jednak często o wiele mniej dokładny.

Korzystając z Solvera, średnio uzyskujemy plansze dokładniejszą o jedno puste pole kosztem czekania o 2 sekundy.

Wnioski:

Łamigłówka Futoushiki okazała się być problemem, który w dość łatwy sposób można rozwiązać lub sprawdzić jego rozwiązywalność za pomocą SOLVERA.

Metaheurystyka, o ile szybciej poradziła sobie z tym problemem dla zadanej temperatury, to co widać po różnicy nawet 6 zerach w wynikach które miały służyć jako przybliżenie optimum, nie okazała się być dobrym rozwiązaniem.

Podejrzewam, że gdybym ustawił czas wyżarzania na dłuższy manipulując lepiej temperaturą początkową lub alfą, metaheurystyka poradziła by sobie o wiele lepiej.

Konieczność wprowadzania kroków takich jak generacja sąsiadów, oraz manipulowania temperaturą sprawia, że rozwiązywanie tego typu problemów w CPLEX jest o wiele wygodniejsze.